Java-Anwendungen mit grafischer Benutzeroberfläche

Bei der Erstellung von Java-Anwendungen mit grafischer Benutzeroberfläche (GUI) verwendet man einige neu objektorientierte Konzepte, auf die im folgenden eingegangen werden soll.

Innere Klassen

Häufig braucht man bei der Programmierung einer Klasse eine Hilfsklasse. Diese ist im Zusammernhang mit der äußeren sinnvoll und soll nach außen hin verborgen sein. Solche Hilfsklassen nennt man *innere Klassen*. Sie werden innerhalb der äußeren Klasse definiert.
Innere Klassen verwendet man häufig bei der Definition von Listener.

Beispiel:

```
public class InnereKlasseDemo{

  static class InnereKlasse{
    public int Eigenschaft=0;
    public void zeige(){
      System.out.println(Eigenschaft);
    }
  }

  public static void main(String args[]){
    InnereKlasse c=new InnereKlasse();
    c.Eigenschaft=2;
    c.zeige();
  }
}
```

In diesem Beispiel ist die Klasse `InnereKlasse` eine innere Klasse der äußeren Klasse `InnereKlasseDemo`. Das `static` vor der Definition von InnereKlasse ist in diesem Fall notwendig, da keine Instanzen, d.h. keine konkreten Objekte, von InnereKlasseDemo erzeugt werden.

Aufgabe: Was wird im obigen Beispiel ausgegeben?

Interfaces

Interface heißt übersetzt Schnittstelle. Interfaces in der Programmierung sind eine Ansammlung von Methodensignaturen und u.U. Konstanten. Das heißt, es wird nur jeweils der Rückgabewert, der Methodenname und die Parameter angegeben. Die Methoden werden an dieser Stelle *nicht definiert*! Damit garantiert man das Vorhandensein der im Interface definierten Methoden. Wenn eine Klasse ein Interface erfüllt, sagt man: „Die Klasse implementiert das Interface.“ Die Klasse muss dann alle Methoden ausführen, die im Interface deklariert sind.

Interfaces werden insbesondere beim Erstellung von Listener verwendet. Die Adapterklassen, die wir beim Applet verwendeten, implementieren alle Methoden der Listener Interfaces, so dass man nur die verwendeten Methoden überschreiben muss.

Beispiel:

```
public class Konto{
  public int Nummer;
  public double Guthaben;

  public void abheben(double betrag){
    if(Guthaben>betrag) Guthaben=Guthaben-betrag;
  }

  public void einzahlen(double betrag){
    Guthaben=Guthaben+betrag;
  }
}

public interface Verzinsbar{
  double zinsenBerechnen();
}

public class Sparkonto extends Konto implements Verzinsbar{
  public double Zinssatz;

  public double zinsenBerechnen(){
    return Guthaben*Zinssatz/100;
  }
}
```

Im Interface Verzinsbar wird eine Methode namens zinsenBerechnen ohne Parameter mit einem Rückgabewert vom Typ double deklariert. Jede Klasse, die dieses Interface erfüllt, muss eine solche Methode besitzen. Wie dies durchgeführt wird, sieht man an der Klasse Sparkonto. Sie erbt („extends“) von der Klasse Konto und implementiert („implements“) das Interface Verzinsbar.
In Java kann jede Klasse nur von *einer anderen Klasse erben*/abgeleitet sein, aber *beliebig viele Interfaces implementieren*.

Java-Applikationen

Neben Applets kann man in Java auch grafische Anwendungen schreiben, die nicht im Webbrowser ablaufen. Diese nennt man Java-Applikationen.
Java-Applikationen haben so wie die ersten Programme wieder eine statische main-Methode, werden aber in der Regel von der Klasse `JFrame` abgeleitet.
Eine einfache funktionierende Java-Applikation sieht so aus:

```
import javax.swing.JFrame;

public class ErsteApplikation extends JFrame{

  // Konstruktor
  public ErsteApplikation(){
    // Titel setzen
    setTitle("Erste Anwendung");  // (1)
    // Größe setzen
    setSize(200,100);      // (2)

    // ganz wichtig:
    // ansonsten kann Anwendung nicht beenden werden
    setDefaultCloseOperation(EXIT_ON_CLOSE); // (3)
  }

  public static void main(String args[]){
    // Objekt von dieser Klasse erzeugen
    ErsteApplikation app=new ErsteApplikation();
```

```
    // Frame anzeigen
    app.show();
  }
}
```

Im Konstruktor wird der Titel und die Größe des Fensters festgelegt, anschließend sorgt man noch dafür, dass wenn das Fenster geschlossen, auch die Anwendung beendet wird.
In der main-Methode erzeugt man ein Objekt der aktuellen Klasse und zeigt dieses mit Hilfe der show-Methode an.
Wenn man Entwicklungsumgebungen wie Forte, NetBeans oder JBuilder verwendet, sieht die einfachste Anwendung ein wenig anders aus, trotzdem enthalten alle Varianten in etwa die gleiche Funktionalität.

Aufgaben:

1. Kommentiere jeweils eine der mit (1), (2), (3) markierten Zeilen mit Hilfe von // aus, kompiliere die Klasse neu und führe das Programm aus. Notiere jeweils die Auswirkungen.
2. Wandle das Mal-Applet in eine Java-Anwendung um.

Layout-Manager

Mit Hilfe von Layout-Managern ordnet Java Kontrollelemente, wie z.B. Knöpfe, unabhängig von der absoluten Größe der beteiligten Objekte an. Man kann damit die Oberfläche also logisch definieren. Die Positionen und Größen bei der beteiligten Elemente werden dabei zur Laufzeit in Abhängigkeit von der Fenstergröße berechnet. Ändert sich die Größe des Containers, z.B. die Größe des Fensters, so ändern sich automatisch die anderen Elemente. Layout-Manager berücksichtigen auch, dass Kontrollelemente von verschiedenen Betriebssystemen unterschiedlich dargestellt werden. Ohne ihre Verwendung können leicht Texte abgeschnitten werden.

Java kennt verschiedene Layout-Manager:

- Beim `BorderLayout` werden die Komponenten im Container an den Rändern angeordnet. Diese werden mit Himmelsrichtungen (NORTH, SOUTH, WEST, EAST) bzw. CENTER bezeichnet.
- Beim `FlowLayout` werden alle Komponenten nacheinander in einer Zeile angeordnet, bis der Rand des Containers erreicht ist. Weitere Komponten werden in der nächsten Zeile hinzugefügt. Durch die Eigenschaften hgap, vgap und alignment kann man das Aussehen beeinflussen.
- Beim `GridLayout` wird der Container in gleich große Zeilen und Spalten aufgeteilt, also wie ein Gitter. Die einzelnen Komponenten werden in das Gitter der Reihe nach gesetzt und füllen dieses am Ende vollständig aus. Dazu werden alle Komponenten auf die gleiche Größe gebracht. Die Anzahl der Spalten und Zeilen wird durch die Eigenschaften columns und rows bestimmt. hgap und vgap legen den Abstand zwischen den Kontrollelementen fest.
- Außerdem gibt es noch das BoxLayout und GridBagLayout.

Aufgabe:

3. Experimentiere mit den verschiedenen Layoutmanagern und ihren Eigenschaften und Komponenten
4. Verwende in der Mal-Applikation als Layout-Manager ein Border-Layout und füge oben („NORTH“) eine Schaltfläche (JButton) und bei CENTER ein JPanel ein.

